モダンアプリケーションアーキテクチャーを活用したCPSソフトウェア開発技術

Use of Modern Application Architecture for Development of CPS Software

舟城　亮一 FUNAKI Ryoichi　　花井　克之 HANAI Katsuyuki

社会インフラを支えるCPS（サイバーフィジカルシステム）のソフトウェア開発では，独立性の高いマイクロサービスと，効率的なデプロイ（実行環境へモジュールを配置すること）を提供するコンテナプラットフォームによって，安定したシステムを実現できる，モダンアプリケーションアーキテクチャーの活用が有効である。

東芝グループは，開発フレームワークやガイドラインの標準化により，モダンアプリケーションアーキテクチャーをソフトウェア開発に活用するための仕組み作りを推進している。また，既存のインフラ系システムのソフトウェア資産を有効活用するモダンアプリケーション化にも取り組んでいる。

In order to develop software for cyber-physical systems (CPS) services, particularly for social infrastructure systems requiring high reliability, there is a need for effective utilization of modern application architecture incorporating highly independent microservices as well as a container platform for the efficient deployment of software.

The Toshiba Group is responding to this need by standardizing the framework of software development and the related guidelines and establishing a software development environment utilizing a modern application architecture. We are also making effective use of accumulated development assets of infrastructure systems as modern applications.

1. まえがき

東芝は，世界有数のCPSテクノロジー企業を目指しており，CPSを支えるソフトウェアを，迅速，柔軟，かつ効率的に開発することが重要である。CPSを支えるソフトウェアに求められる要件については，この特集のトレンドで述べた（この特集のp.2–6参照）。

ここでは，まず，俊敏さと継続的な進化を実現する開発手法であるアジャイル開発や，開発と運用を組み合わせて開発期間を短縮するDevOps, CI/CD（継続的インテグレーション／継続的デリバリー）といった新しいソフトウェア開発・運用手法について，アーキテクチャーのパラダイムシフトの観点から整理して述べる。そして，これらを踏まえた，東芝グループにおけるモダンアプリケーションアーキテクチャー導入の取り組みの概要を述べる。

2. CPSに必要なアーキテクチャー

一つのシステムを，複数のアジャイル開発チームが並行して，俊敏かつ自律的にCI/CDで進化させるためには，そもそもシステムは常に安定状態である必要がある。安定状態でないシステムを，アジャイルに進化させることはできない。ここでは，システムをアジャイルに進化させるのに必要な，システムの安定性を実現する技術について述べる。

CPSは社会インフラを支えるシステムであり，高い安定性が求められる。安定したシステムとは，一過性又は継続的な高負荷や，機器やネットワークの障害，予想していなかったユーザー操作といったストレス環境においても，システムが回復力を持ち，所定の処理を遂行できることを意味する[(1)]。

一般に，システムに対するストレスは，システムを構成しているプロセスのメモリーリークやスレッド障害などを引き起こし，最終的にシステム停止につながることが知られている。ここで，鋼板に加わるストレスと亀裂の発生に例えて，システム障害への対処法を説明する。鋼板にストレスを加えた際の亀裂は，加速度的に鋼板全体に伝搬してしまう。そのため，物理的な世界の設計においては，隔壁を用いることで，万一障害が発生しても，亀裂の伝搬を食い止め，被害が全体に及ばないようにしている。例えば，船の隔壁は損傷の範囲を局所化し，かつ水の浸入を阻止することで沈没を防ぎ，自動車の隔壁は，事故による衝突の際に，人間が生き残るための空間を作る。

ソフトウェアも同様である。近年，亀裂の伝搬を食い止め，システムの安定を実現するアーキテクチャーパターンとして，隔壁モデル，タイムアウト，サーキットブレーカーといった手法が提唱されている（**図1**）。

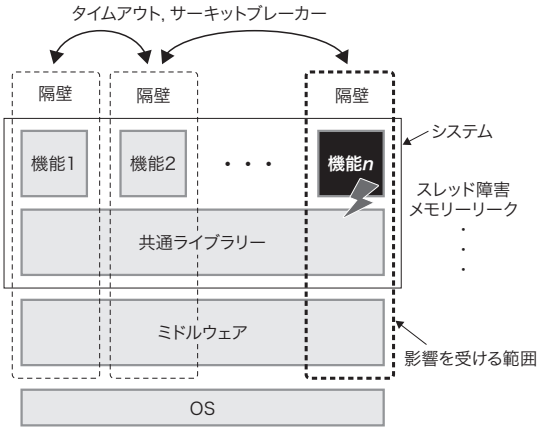

図1. 隔壁モデルの例

システムが，機能ごとに隔壁で区切られており，どれか一つの機能に障害が発生しても，ほかの機能にそれを伝搬させない。

Bulkhead model to prevent propagation of failure to other functions

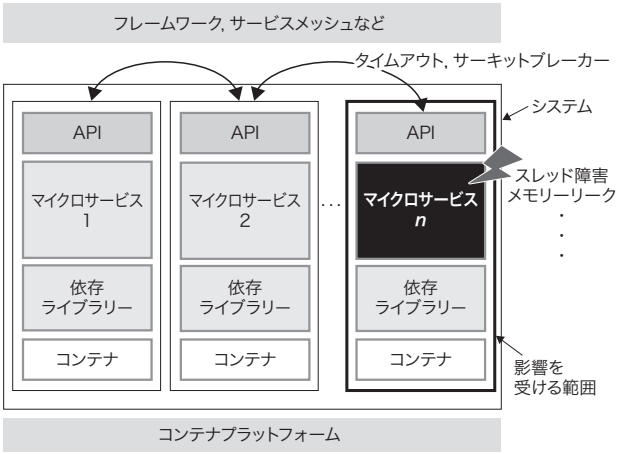

図2. モダンアプリケーションの例

各マイクロサービスがコンテナによりリソース分割されており，障害をほかのマイクロサービスに伝搬させない。

Architecture of modern application

隔壁モデルとは，船や自動車のように，システムに幾つもの隔壁を作り，ストレスによる障害の伝搬を食い止めるという設計思想である。隔壁はプロセスを構成する機能と機能を隔てるものとして考える。

タイムアウトとサーキットブレーカーは，ある機能が万一障害を起こしても呼び出し元を守る仕組みとして重要である。タイムアウトは，相手が無応答の際でもタイムアウトで戻ること，サーキットブレーカーは，単位時間当たりのエラーリターン数がしきい値を超えたら，それ以上その機能を呼び出さないことで，システムの稼働を継続させる。

このような構造のシステムを設計することで，障害の発生を局所化し，更に部分的な障害を復旧させることができれば，システムは常に安定して稼働できる。システム全体を稼働させたままでも一部を変更する作業も可能となる。

3. モダンアプリケーション

CPSの実装においては，2章で述べたような最近の設計思想やアーキテクチャーを取り込んでいく必要がある。近年，柔軟なクラウドシステムの特長を生かした信頼性の高いアプリケーションの総称として，クラウドネイティブ，モダンアプリケーションといったキーワードが使われることが多くなった。

3.1 モダンアプリケーションのアーキテクチャー

図2に，モダンアプリケーションのアーキテクチャーを示す。モダンアプリケーションの明確な定義はないが，システムを，マイクロサービスといった小規模なサービス志向プロセスの組み合わせで構成し，Dockerなどに代表されるコンテナ技術を活用してデプロイの効率化が図られていることが特長である。マイクロサービス同士は，REST API（Representational State Transfer Application Programming Interface）(注1)に代表されるWebプロトコルによるAPIで連携され，サービスメッシュ(注2)やマイクロサービスに対応するフレームワークで，タイムアウトやサーキットブレーカーが実装される。

ここで，図1と図2は非常に類似していることが分かる。図1で提唱している機能と機能を隔てる隔壁を，図2では機能ごとのマイクロサービスとコンテナで実現している。コンテナにより，マイクロサービスごとに，使用できるCPUやメモリーなどのリソースが限定されるため，たとえ暴走やメモリーリークを起こしたとしても，ほかのマイクロサービスには全く影響を与えない。

これは，モダンアプリケーションが，極めて忠実に安定システムのアーキテクチャーパターンを構成しており，マイクロサービス単位で，回復力を実装できることを意味している。更には，マイクロサービスの独立性により，機能ごとの更新や改良が可能になる。これが，モダンアプリケーションが，自律的に分散したアジャイルチームによる開発を可能にした理由である。

コンテナは，マイクロサービスの隔壁を作るだけでなく，デプロイに際しても有益である。コンテナは，マイクロサービスが稼働するために必要となる環境（ライブラリー，OS（基本ソフトウェア）コール，ミドルウェア，パラメーター設定な

（注1） 標準的なWeb技術を用い，HTTP（Hypertext Transfer Protocol）メソッドでリソースの操作が指定できる呼び出しインターフェース。
（注2） マイクロサービスを一括管理する仕組み。

ど）を全てパッケージ化し，動作前に展開する方式を採っている。これは，CI/CDを自動実行する上で，非常に重要なブレークスルーとなった。

このように，モダンアプリケーションは，マイクロサービスといった小規模なサービス志向プロセスによる分散システムを構成しており，それらが全体で安定しているため，自律的に分散したアジャイルチームによる開発や，DevOps，CI/CDなどを実装しやすいアーキテクチャーになっている。その結果，俊敏な開発やシステムの継続的な進化が可能となり，CPSにとって重要なビジネス上の価値創出を実現できる。

3.2　モダンアプリケーション開発時のルール

モダンアプリケーションのアーキテクチャーを採用すれば，誰もがそのメリットを享受できるかというと，そうではない。モダンアプリケーションを書くプログラマー自らが，享受するメリットを意識してコーディングする必要がある。

例えば，システムの回復力を高めるためには，障害を起こしたマイクロサービスは直ちに再起動することでサービスを復旧できなければならないし，大規模なワークロードに対応するためには，マイクロサービスは分散処理及びスケールアウトできなければならない。安定したCI/CDを実現するためには，開発，ステージング(注3)，及びプロダクト環境もできるだけ一致させなければならない。更に，設計段階から，アプリケーション回復性の実装，及び稼働状態や，メトリック測定，出力ログなどをモニターできる仕組みを考えておくことも必要である。

こういった，モダンアプリケーションを書くプログラマーが守るべきルールについては，既に各種文献や書籍など[2]で公開されており，CPSの開発においてもそれら一般化されたメソドロジーに準拠していくことが推奨されている。

4.　モダンアプリケーション活用における東芝グループの取り組み

4.1　マイクロサービス開発のための仕組み

マイクロサービスとコンテナプラットフォームによるモダンアプリケーションは，アジャイル開発やCI/CDの実現に大きく貢献した。特に，コンテナプラットフォームは，今やエッジからオンプレミス，クラウドシステムまで採用される共通プラットフォームとなっている。

3.2節で述べたとおり，新しいアーキテクチャーを採用しただけでメリットを享受できるわけではなく，マイクロサービスの特長の一つである“障害のための設計”が必要となる。ほかにも，分散処理や分散データのための設計も必要となる[3]。企業が，モダンアプリケーションを実装するためには，文献などで紹介されている概念を学ぶだけでは，十分ではない。

このような背景から，東芝グループは，モダンアプリケーションの活用を目的として，マイクロサービス開発を行うためのフレームワークとアーキテクチャーパターンの標準化，各パブリッククラウドにおけるコンテナプラットフォーム構築方針など実行基盤の標準化，及びアプリケーション稼働状況やメトリック情報などの運用ガイドラインの標準化を行っている。それらをマイクロサービス開発技術としてアプリケーション開発現場に展開することで，効率的なマイクロサービス開発と安定したサービス運用を実現するための仕組み作りを進めてきた。

また，その効果を検証するために，東芝グループのサービス・ソリューション開発で実際に利用し，その経験を標準化作業へフィードバックしている。

4.2　既存システムのITモダナイゼーション

最後に，モダンアプリケーションによるCPSの事例として，APIラッピングによる既存システムのIT（情報技術）モダナイゼーションについて述べる。

現在，社会インフラ系のシステムをはじめとして，モダンアプリケーションのアーキテクチャーが提唱される以前に設計されたシステムが，多数存在している。そのようなシステムが大幅な改造なしにモダンアプリケーションのメリットを享受できるようにする手法として，APIラッピングがある。これは，既存システムの上位にモダンアプリケーションと同等のAPIを実装したサービスアプリケーションを配置し，既存システムをラッピングする方法であり，その結果，システム全体をモダンアプリケーションとして扱うことができる（**図３**）。

この手法を活用することで，CPSのフィジカル部分において，東芝グループが過去に作成し，今なお現役で稼働し続けている膨大な社会インフラのソフトウェアアセットを効率良く活用できる。拡張性に優れるモダンアプリケーションのアーキテクチャーを採用したサイバーシステムと組み合わせることで，優れたCPSを提供することが可能になる。

このように，既存のシステムをモダンアプリケーションに移行することをITモダナイゼーションと呼ぶ。東芝デジタルソリューションズ（株）は，数年前から様々なITモダナイゼーション手法を，情報系システムのモバイル対応や，SaaS（Software as a Service）連携などの実システムに適用し，実績を積んできた。

近年は，このような経験を生かして，社会インフラシステムとモダンアプリケーションの連携事例も多くなっており，そのノウハウは，CPS開発にも大いに生かされている。

（注3）ソフトウェアリリースにおいて，プロダクト環境（本番環境）に公開する前に，最終的な動作確認を行うフェーズ。

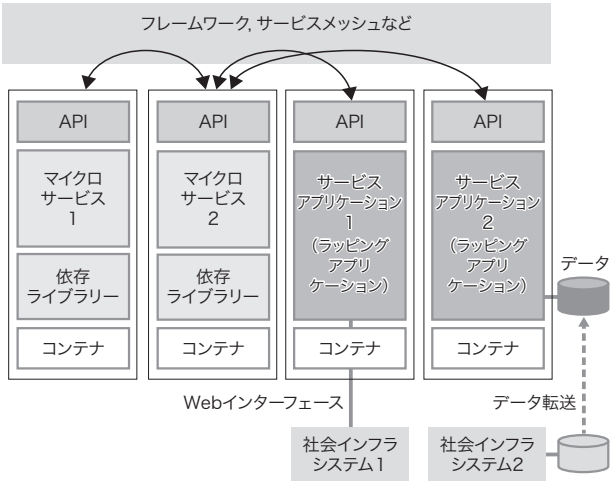

図3. 社会インフラシステムのAPIラッピングの例

従来技術で構築された社会インフラシステムをAPIでラッピングし，システム全体をモダンアプリケーション化する。

Example of application programming interface (API) wrapping of social infrastructure systems

5. あとがき

ここではモダンアプリケーションアーキテクチャーの特長を，マイクロサービスとコンテナプラットフォームによる隔壁モデルの実現の観点から述べた。

東芝グループは，今後もモダンアプリケーションに対応したマイクロサービス開発を支援するマイクロサービス開発技術の継続強化を図り，開発スピードと品質を両立するCPSソフトウェア生産技術を確立していく。

文　献

(1) Nygard, M. T. Release It!: Design and Deploy Production-Ready Software. 2nd ed., Pragmatic Bookshelf, 2018, 376p.
(2) Wiggins, A. The Twelve-Factor App. 2017, 28p. <https://12factor.net/>, (accessed 2020-06-15).
(3) Lewis, J. ; Fowler, M. Microservices. <https://martinfowler.com/articles/microservices.html>, (accessed 2020-06-15).

舟城　亮一　FUNAKI Ryoichi
ユアサ商事（株）グローイング戦略本部
東芝デジタルソリューションズ（株）を経て，2020年8月から現職
Yuasa Trading Co.,Ltd.

花井　克之　HANAI Katsuyuki
東芝デジタルソリューションズ（株）ソフトウェアシステム技術開発センター IT モダナイゼーション推進部
情報処理学会会員
Toshiba Digital Solutions Corp.